North *Complex Adaptive Systems Modeling* 2014, 2:3
http://www.casmodeling.com/content/2/1/3

Complex Adaptive Systems Modeling
a SpringerOpen Journal

RESEARCH Open Access

A theoretical formalism for analyzing agent-based models

Michael J North[1,2]

Correspondence: north@anl.gov
[1]Decision and Information Sciences Division, Argonne National Laboratory, 9700 South Cass Avenue, Lemont, IL 60439, USA
[2]University of Chicago, 5801 South Ellis Avenue, Chicago, IL 60637, USA

Abstract

Purpose: Following Holland, complex adaptive systems (CASs) are collections of interacting, autonomous, learning decision makers embedded in an interactive environment. Modeling CASs is challenging for a variety of reasons including the presence of heterogeneity, spatial relationships, nonlinearity, and, of course, adaptation. The challenges of modeling CASs can largely be overcome by using the individual-level focus of agent-based modeling. Agent-based modeling has been used successfully to model CASs in many disciplines. Many of these models were implemented using agent-based modeling software such as Swarm, Repast 3, Repast Simphony, Repast for High-Performance Computing, MASON, NetLogo, or StarLogo. All of these options use modular imperative architectures with factored agents, spaces, a scheduler, logs, and an interface. Many custom agent-based models also use this kind of architecture. This paper's contribution is to introduce and apply a theoretical formalism for analyzing modular imperative agent-based models of CASs. This paper includes an analysis of three example models to show how the formalism is useful for predicting the execution time and space requirements for representations of common CASs.

Method: The paper details the formalism and then uses it to prove several new findings about modular imperative agent-based models.

Results: It is proven that the asymptotic time and space performance of modular imperative agent-based modeling studies is computationally optimal for a common class of problems. Here 'optimal' means that no other technique can solve the same problem computationally using less asymptotic time or space. Modular imperative agent-based models are shown to be universal models, subject to the correctness of the Church-Turing thesis. Several other results are also proven about the time and space performance of modular imperative agent-based models. The formalism is then used to predict the performance of three models and the results are found to compare closely to the measured performance.

Conclusions: This paper's contribution is to introduce, analyze, and apply a theoretical formalism for proving findings about agent-based models with modular agent scheduler architectures. Given that this kind of modeling is both computationally optimal and a natural structural match for many modeling problems, it follows that it is the best modeling method for such problems.

Keywords: Agent-based modeling; Computational complexity theory; Random access stored-program machine

Background

Introduction

Complex adaptive systems (CASs) are collections of interacting, autonomous, learning decision makers embedded in an interactive environment (Holland 1992; Holland 1999; Holland 2006). CASs are common in both nature and society. Examples include ecosystems composed of organisms, industrial supply chains consisting of companies, social networks of formed by people, and even the human body's multitudinous cells. The contributing decision makers in each of these CAS, and all others, have properties and behaviors. The decision makers interact with and influence each other; learn from their experiences; and change their behaviors so they are better suited to their environment.

Modeling CASs is challenging for several reasons. The details matter, so averages often fail to properly represent these systems. Heterogeneity is the norm, so decision makers must be modeled individually. Adaptation is expected, so models must be highly dynamic. Space is central, so models must be intensively spatial. Scale matters, so models must often include large numbers of decision makers. Nonlinearity is routine, so models must track specific decision maker attributes against relevant thresholds. As an example, consider human tissue growth at the cellular level. In this situation, averages over the number of cells and extracellular materials do not take into account the unique, heterogeneous situation of each cell. Cells adapt to their circumstances and substantially change their behavior in nonlinear ways. The relative locations of cells and extracellular materials are critical for predicting long-term outcomes. Furthermore, scale matters since large numbers of cells are needed to form tissues. Consider further an example of tissue engineering where human tissues such as bone are grown in controlled environments. Now, there are nonlinear and even adaptive environmental inputs. If vascularization (e.g., blood vessel formation) is one of the goals then appropriate growth factors may be periodically added to the growth media. If growth optimization is the goal then additional factors and nutrients may also be included.

The challenges of modeling CAS such as those for tissue growth and engineering can largely be overcome by using the individual-level focus of agent-based modeling (ABM) (Bonabeau 2002; Macal and North 2010; North and Macal 2007). By modeling CAS decision makers as individual agents, the full effects of the diversity that exists among the decision makers with respect to their attributes and behaviors can be observed as they give rise to the dynamic behavior of the system as a whole. Like CAS decision makers, ABM agents have properties and behaviors, interact with and influence each other, learn from their experiences, and adapt their behaviors so they are better suited to their environment. Agent-based modeling has been used successfully to model CAS in many disciplines, including archaeology, biology, ecology, supply chains, consumer market analysis, military planning, and economics (North and Macal 2007; North and Macal 2009). Many of these models were implemented using agent-based modeling software such as Swarm (Accessed 2014), Repast 3 (Accessed 2014), Repast Simphony (Accessed 2014), Repast for High-Performance Computing (Accessed 2014), MASON (Accessed 2014), NetLogo (Wilensky 2014), and StarLogo (Accessed 2014). All of these options use modular imperative agent-based modeling architectures. Many custom agent-based models also use this kind of architecture. Three examples are reviewed later in the section on example models. These examples to show how the formalism introduced in this paper is useful for predicting the execution time and space requirements for representations of common CASs.

There are many possible theoretical foundations for computational agent-based modeling. The value of each candidate depends on the questions to be answered by the theoretical approach. For example, the theoretical foundation in North (2012) focuses on how information is actually processed in agent-based models. Other candidates are reviewed in the section on related work. This paper extends the implementation-oriented theoretical foundation in North (2012) and applies it to study a variety of new questions.

This paper's contribution is to introduce and apply a theoretical formalism for analyzing agent-based models. This paper details the formalism and then uses it to prove several new findings about agent-based models. It is proven that the asymptotic time and space performance of agent-based modeling studies is computationally optimal for common classes of problems. Here 'optimal' means that no other technique can solve the same problem computationally using less asymptotic time or space. Asymptotic approaches are likely to become increasing attractive as the sizes of agent-based models grow (Parker and Epstein 2011; Murphy 2011; Collier and North 2012). Agent-based models are also shown to be universal models, subject to the correctness of the Church-Turing thesis (Kleene 1943). Here 'universal' means that any computational model can be expressed as an agent-based model. Several other results about agent-based model time and space performance are also proven. The formalism is then used to predict the performance of three agent-based models and the results are found to compare closely to the measured performance.

The formalism contributes several things to the modeling of CAS and is therefore needed by CAS researchers from a range of disciplines. First, the formalism provides fundamental results about the time and space performance of one of the most common ways to investigate CASs, namely modular imperative agent-based models. Second, the formalism provides a way to predict the runtime and memory required to execute modular imperative agent-based models of CASs. This will become increasingly important as the number of agents commonly represented in models rises. Third, the formalism provides a straightforward way for CAS model developers to compare the time and space performance of modular imperative CAS agent-based models to that of traditional approaches such as optimization. Fourth, the formalism anchors modular imperative agent-based models, and their associated CASs, within the algorithmic hierarchy of traditional computer science complexity theory.

Related work

There is a range of published work that uses formal systems to prove selected properties of agent-based models or related kinds of models. This work includes the use of equation-based modeling, game theory, discrete systems, classifier agents, and an earlier random access stored-program (RASP) machine-based formalism. Each of these candidates is reviewed in the next sections.

Equation-based modeling

Parunak et al. (1998) compare agent-based modeling and equation-based modeling. They conclude in part that agent-based modeling is best for spatial problems and individual choices, while equation-based modeling is best for geographically concentrated problems driven by well-defined mathematical rules. Specific equation-based formalisms for analyzing simulations include partial recursive functions, difference equations, and dynamical systems.

Partial recursive functions

Epstein (2007) invalidates the commonly cited dichotomy between computational agent-based models and equation-based models by proving, albeit informally, that all computational agent-based models can be expressed using partial recursive functions. Epstein begins his proof by observing that because any standard computer program is effectively Turing complete, then programs for computational agent-based models must be so as well. Epstein then notes that there is an equivalent partial recursive function for every Turing machine. Therefore, there are sets of partial recursive functions for every computational agent-based model. These partial recursive functions are equations that fully specify any given computational agent-based model. Epstein also uses the partial recursive functions approach to argue informally that computational agent-based models, particularly those in the social sciences, are deductive in nature.

Considering the partial recursive functions approach further, Epstein (2007) discusses how unnatural or "unrecognizable" this expression of an agent-based model is likely to be for most agent-based modelers and how far this formalism is from real implementations. This suggests that partial recursive functions are not presently a suitable implementation-oriented theoretical foundation for agent-based modeling.

Difference equations

Leombruni and Richiardi (2005) introduce an equation-based formalism that can describe both agent-based and analytical models. Their formalism is intended to bridge the conceptual gap between equation-based economics and agent-based modeling.

Leombruni and Richiardi (2005) define agent attributes using vectors of state variables and then employ time-indexed difference equations to specify how each agent's state changes from one time step to the next. Naturally, analytical models that can be reduced to difference equations can also be represented with their formalism. Once they define their formalism, they then consider the question of how to estimate the relevant parameters in the difference equations.

Leombruni and Richiardi (2005)'s formalism has the advantage of including both agent-based and analytical models. Unfortunately, difference equations can be an extremely awkward way to specify the behavior of agents with complicated algorithmic behavior. Furthermore, they do not use their formalism to prove properties of either kind of model or show how this could be done.

Algebraic dynamical systems

Hinkelmann et al. (2011) extend Grimm et al.'s (2006) Overview, Design Concepts, and Details (ODD) protocol by adding an algebraic field to describe agent behavior. First we review ODD then consider Hinkelmann et al.'s (2011) extension of it.

ODD describes models using a three-part approach involving an overview, concepts, and details. The model overview includes a statement of the model's intent, a description of the main variables, and a discussion of the agent activities. The design concepts include a discussion of the foundations of the model. The details include the initial setup configuration, input value definitions, and descriptions of any embedded models.

Hinkelmann et al.'s (2011) ODD extension maps each agent's state into a set of state variables. A set of time-indexed polynomial discrete dynamical equations that update the values of each agent's state variables are then defined in such a way as to allow the

state variables to form an algebraic field. They then use the rich body of existing algebraic techniques to prove system-level properties of two simple demonstration models.

Hinkelmann et al.'s (2011) approach has the advantage of offering a straightforward way to prove properties of agent-based models while leveraging a large body of existing research. Unfortunately, expressing complicated agent behaviors using time-indexed polynomial discrete dynamical equations can become prohibitively complex. Also, Hinkelmann et al.'s (2011) note that the algebraic techniques they seek to leverage sometimes have serious weakness of their own, such as an inability to scale to realistic model sizes.

Reaction–diffusion dynamical systems

Dynamical systems are sets of equations that specify a path through a space over time (Smale 1967). Selected agent-based models can be expressed as dynamical systems models or can be approximated by them. For example, Dorofeenko and Shorish (2002) used this approach to study a variation of the prisoner's dilemma game. It is interesting to note that even though the prisoner's dilemma is a classic game theoretic model, Dorofeenko and Shorish did not choose analytical or algorithmic game theory as the primary method for their study.

Dorofeenko and Shorish (2002) map Epstein's (1998) demographic prisoner's dilemma game to a reaction–diffusion dynamical system metamodel. Epstein's original game used a 30×30 square lattice populated by reproducing agents with fixed prisoner's dilemma strategies. Dorofeenko and Shorish investigated a one-dimensional version of the game and proved that certain conditions produce regions of long-term cooperation. Their findings are interesting, but are limited to the specific model that they are investigating. In particular, they do not provide a method for generalizing their findings or applying dynamical system analysis to a wider range of agent-based models. This absence illustrates the general conclusion that dynamical systems have limited potential as a theoretical foundation for agent-based modeling, as there is no known general mapping between computational agent-based models and dynamical systems models.

Game theory

Game theory is a mathematical approach for studying multiparty decision problems in interactive environments (Myerson 1991). Some of the findings that are the most relevant for this paper are that many games have at least one stable outcome (i.e., Nash equilibrium) and that many games are nondeterministic polynomial time (NP)-complete (Nisan et al. 2007). Nisan et al. (2007) show that games with exactly one Nash equilibrium are in a lower complexity class than are NP-complete problems, whereas games with two or more Nash equilibriums are NP-complete.

Game theory can be used to prove properties about agent-based models to the extent that such models can be expressed as games. There are several approaches to solving game theoretic formulations including analytic game theory and algorithmic game theory. Each will be considered separately.

Analytic game theory (Nisan et al. 2007) applies mathematical proof techniques to deduce facts about game theoretic formulations. The previously mentioned proof that certain games are NP-complete is an example. Unfortunately, only a few kinds of games are amenable to analytical proof techniques (Nisan et al. 2007). Therefore, most practical problems do not have a corresponding analytically solvable game. This constraint

prevents analytic game theory from being a useful general theoretical foundation for computational agent-based modeling.

Algorithmic game theory (Nisan et al. 2007) uses computational tools to solve game theoretic problems. Its computational nature allows it to address a much wider range of cases than does analytic game theory. However, this added flexibility often comes at the expense of theoretical generality.

There are several major differentiators between algorithmic game theory and agent-based modeling. Algorithmic game theory tends to focus on specific types of outcomes, such as stable equilibrium or various kinds of oscillatory dynamics (Nisan et al. 2007). Agent-based modeling tends to address much broader outcomes (North and Macal 2007). Algorithmic game theory uses extended game formulations to specify models, whereas agent-based modeling typically uses more scalable software engineering design tools. Unlike algorithmic game theory formulations, agent-based modeling software is often designed to represent detailed geographies, rich interaction topologies, nuanced agent learning, and complex agent behaviors. Therefore, algorithmic game theory is not an appropriate theoretical foundation for agent-based modeling because of the limited kinds of outcomes that are represented, the constraints on scaling, and the substantial practical differences in how models are specified.

Discrete systems

Quite a few different approaches to representing agent-based models using discrete systems have been published. These approaches include cellular automata and the Discrete Event System Specification (DEVS). These approaches will be considered in the following sections.

Cellular automata

Cellular automata (Christos and Lafortune 2008) are grids of cells, each of which has a state transition diagram. Uniform cellular automata have a single state transition diagram for every cell. Non-uniform cellular automata can have state transition diagrams that vary by cell. Brown et al. (2005) discuss cellular automata in relation to spatial agent-based modeling. They observe that cellular automata tend to yield Eulerian (i.e., field-oriented) models versus Langrangian (i.e., individual-oriented) representations. Here we conclude that this limits cellular automata as a foundation for proving properties about agent-based models, since many are domains are naturally Langrangian.

DEVS

DEVS is one of the best developed theoretical formalisms for analyzing complicated simulations. DEVS describes simulations as hierarchical Moore machines with timing information added to the state transitions (Zeigler et al. 2000). Classic DEVS is the core abstraction. It is a structure that consists of the following:

- A set of external events that can be input to the DEVS,
- A sequence set of states,
- A set of internal results that can be output from the DEVS,
- An internal transition function that specifies how timeouts change states,
- An external transition function that specifies how inputs change states,

- A time advance function that specifies how long before each state times out, and
- An output function that specifies the results for each state.

Coupled DEVS extends classic DEVS by allowing hierarchical nesting of the Moore machines. An example of a hierarchical DEVS implementation is described by Bolduc and Vangheluwe (2002). Extensions that add other functionality to classical DEVS and restrictions that limit the functionality have also been studied.

DEVS provides a theoretical foundation for analyzing some critical computational properties of agent-based models such as decidability. If the Church-Turing thesis is correct, then any computing system as complex as a Moore machine can be used to calculate any function. Therefore, in principle, DEVS can represent any computable model. Nonetheless, this finding by itself does not mean that DEVS is an ideal theoretical foundation for agent-based modeling because the same conclusion can be reached about any Turing-complete system.

Unfortunately for agent-based modeling, DEVS imposes an awkward boundary between state and behavior that makes it difficult to represent dynamically generated agent behaviors. The DEVS boundary also does not correspond to common agent-based model architectures. In addition to dynamically generated behaviors, agent-based models often have state structures that are a dynamic function of the events in the models. Dynamic Structuring DEVS (Shang and Wainer 2006) attempts to address these weaknesses, but it also adds new constraints that similarly limit its utility.

In addition, DEVS lacks important constraints commonly found in agent-based models. Agent-based models have extremely wide-ranging properties, yet exhibit characteristic forms of state storage and agent behavior. These conventions place specific theoretical limitations on the computational properties of agent-based models. DEVS does not include these conventions. An appropriate theoretical foundation should represent these conventions and allow analysts to reason about them rigorously. These agent-based modeling conventions are detailed in later sections.

Classifier agents

Holland (2006) discusses the modeling of complex adaptive systems using a variety of techniques including agent-based modeling. He briefly reviews the capabilities of control theory, economic modeling, biological cell modeling, and game theory. Holland concludes that although the formalisms he has reviewed have unique strengths, they all face the problem of relying on differential calculus in general and on partial differential equations in particular. He then argues that the usefulness of partial differential equations is undermined by the inherent nonlinearity of the domains being modeled. For Holland, the answer is to develop exploratory computer-based models that have agent rules defined using individual classifier systems. Holland's approach does not include enough explicit simulation engine details to derive time or space performance bounds. Furthermore, his approach is a special case of the formalism presented in this paper because it can be implemented using this formalism. Unlike this paper, Holland's paper does not include time or space bounds derived using his recommended approach.

Formal methods

Niazi and Hussain (2010) use the Z specification language to represent an agent-based model of wireless sensor networks. Z is an International Standards Organization standard that specifies computations using set theory and predicate calculus operating on typed expressions (Spivey 1992). Z and its derivatives offer correctness-proving tools and model-based checking capabilities. Z specifications are often detailed enough to be translated into executable software.

Niazi and Hussain (2010) show how to use Z to specify a wireless sensor network model by defining an interlocking set of Z schemas. They report that the resulting specification is useful for demonstrating or proving the correctness of the wireless sensor network model relative to well-defined requirements. Also, they show that the Z specification provides a way to document detailed expectations for model behavior. This matches well with two of Z's purposes, namely system documentation; and system verification and validation. However, the Z language does not specifically address model execution time or space performance analysis. For example, Niazi and Hussain (2010) do not use their Z specification to predict the execution time or space performance of their wireless sensor network model.

Earlier RASP formalisms

North (2012) introduces a RASP-based formalism for the analysis of agent-based model performance and uses this formalism to prove that the total asymptotic time and space performance of a specific Mars Rover model is computationally optimal. The current paper substantially extends North (2012) in several ways. First, the formalism presented in this paper is both different and more general than that of North (2012). The formalism in this paper includes a discrete event scheduling option, a wider range of interaction spaces, a discussion of user interfaces, analysis of parallel execution, and provisions for modeling studies. Second, broadly applicable computational optimality conditions are proven in this paper. Third, several other results about agent-based model time and space performance are also proven. Fourth, this paper provides performance analysis examples for three real models and compares these results to the theoretical predictions.

Methods

Common features of agent-based modeling libraries

An implementation-oriented theoretical foundation for computational agent-based modeling should take into account the conventions commonly seen in applied modeling. This is not to say that current agent-based modeling efforts represent the expected final form of computational agent-based modeling research. Agent-based modeling is younger than techniques such as game theory. It is anticipated that agent-based modeling will evolve rapidly in the future. As this growth proceeds, it is hoped that agent-based modeling theory and practice will advance together. This paper is intended to contribute to this evolution.

Seven free and open-source agent-based modeling libraries and environments with published descriptions will be used to illustrate the common structural conventions of agent-based models. The examples are Swarm, Repast 3, Repast Simphony, Repast for

High-Performance Computing (HPC), MASON, NetLogo, and StarLogo. Each of these options has a modular imperative agent-based modeling architecture.

Swarm is an agent-based modeling library with both Objective-C and Java bindings (Minar et al. 1996). Repast 3 (North et al. 2006) is a family of three agent-based modeling libraries, two in Java and one in C#, each with architectures inspired by Swarm. Repast Simphony (North et al. 2013) is a Java-based library and environment that generalizes Repast 3. Repast HPC (Collier and North 2012) is a C++ Message Passing Interface agent-based modeling library designed to run Repast Simphony's core feature set efficiently on supercomputers.

MASON is an agent-based modeling library that seeks to provide a focused set of core services that can be easily used with other libraries (Luke et al. 2005). Its structure is related to that of Swarm.

NetLogo (Wilensky 1999) is an educational agent-based simulation environment that uses a modified version of the Logo programming language (Harvey 1997). Repast Simphony and Repast HPC include a ReLogo module and environment inspired by Logo in general and NetLogo in particular. StarLogo (Resnick 1994) is another Logo environment related to NetLogo.

When considering the commonalities between these libraries and environment, a short list of features emerges. These features include agents, a scheduler, interaction spaces, a random number source, logging, and a user interface. These features are discussed in detail in the sections that follow. It should be noted that these features are significantly different than those suggested by the previously discussed DEVS formalism.

Agents

Agents are the decision makers in all of the selected libraries and environments. Methods for representing agents vary. For example, the simulation systems discussed above are all object-oriented except for NetLogo and StarLogo. At a minimum, agents have some type of a unique identifier (e.g., index code or memory reference), behaviors that can be activated, and attributes that can be modified. Every agent can have unique behaviors and attributes, although agents are often grouped into types, classes, or breeds. Some libraries or environments offer additional features, such as dynamically modifiable behaviors.

Initialization of agents is completed using a bootstrap module that is called before the model begins executing. Some libraries or environments also offer persistent storage using text files, extensible markup language (XML) files, binary files, databases, or clouds.

Scheduler

The scheduler is responsible for representing the flow of time in a simulation. Scheduling can be time stepped or can use discrete events.

Time-stepped agents activate once for each scheduler time increment. The time counter normally increases in fixed increments of one unit for each step. Schedulers often implement time-step scheduling by repeatedly scanning through a list of agents. This list may be constant or may be randomized over time to allow the agent activation order to vary.

With discrete event scheduling, each agent action occurs at a unique time. Time advances from event to event. Some systems allow multiple events at a given time.

These pseudo-parallel systems usually execute simultaneous events at a given time in a randomized sequential order.

Schedulers often implement discrete event scheduling by maintaining a list of pending, time-stamped agent behaviors sorted by time of activation. The next behavior is activated whenever a running behavior is completed. If any ties occur, they are often broken with random draws as previously discussed.

Interaction spaces

Several different kinds of agent interaction spaces are usually available. All of the previously listed libraries and environments offer several basic interaction spaces, except for StarLogo, which does not have networks:

- A finite, one- (or more) dimensional integer lattice with one computable real value for each lattice point provides a simple way to represent scalar fields. The number of dimensions for each scalar field is fixed when the grid is created.
- A finite, one- (or more) dimensional integer lattice with multiple occupancy and computable real number coordinates for agents provides a simple representation of physical space. The computable real numbered coordinates allow each agent to take on any location within the range supported by the grid. The integer lattice allows agents to find other agents who have similar coordinates (i.e., those agents with the same location when their coordinates are truncated to an integer). As with scalar fields, the number of dimensions for each grid is fixed when the grid is created. This entity will be called a "grid."
- A basic network stored as a weighted graph between agents provides a way to represent connections between agents. Both directed and undirected graphs are supported.

Each model can have zero or more of each kind of interaction space. Generally, the number of interaction spaces is a small constant relative to the number of agents. By default, every agent in each model is present somewhere in each of the model's interaction spaces.

Random number streams

Random number streams are used to generate reproducible sequences of pseudorandom numbers. Several probability distributions are usually provided, including uniform random sources.

Logging

Logging provides a mechanism for recording values within an executing model for later analysis. Data collection is usually performed at regular intervals during model execution both for the overall model and for sets of agents. Data collection is usually performed automatically based on user input. User-programmed logging will be used in this paper.

User interface

Agent-based models usually run in one of two modes. The first is an interactive mode that presents the user with a graphical interface. The second is a 'headless' batch mode

that reads input files, the runs model, and then stores the results into output files. Databases, data warehouses, or clouds may also be used in place of files.

Agent-based models are often run in interactive mode during model development and scenario creation. Graphical user interfaces (GUIs) for agent-based models are usually composed of a main window containing a nested set of subwindows. The main window is called the 'frame'. Sometimes a frame is not used and the subwindows appear independently. In either case, the subwindows contain various simulation controls, one or more views of the spaces, and one or more displays of the properties of selected individual agents. The agent property subwindows are called 'probes'. The properties often dynamically update as the underlying model executes. An overview schematic is shown in Figure 1.

Displaying a GUI often consumes a large amount of computing time and space. Most of these resources are not needed during batch runs. Therefore, batch mode is commonly used to generate large numbers of runs once the model and scenarios are finalized. Batch mode is also typically used for individual model runs with large numbers of agents. Batch mode will be the main focus of this paper because asymptotic performance analysis of agent-based models implies large numbers of agents.

Abstract machine for modular imperative agent-based modeling

This section provides an abstract machine specification for the modular imperative agent-based modeling architecture identified in the previous section. In this paper, we use the RASP architecture for our abstract machine. In keeping with the implementation-oriented theme discussed earlier, the RASP architecture was selected because it has a closer

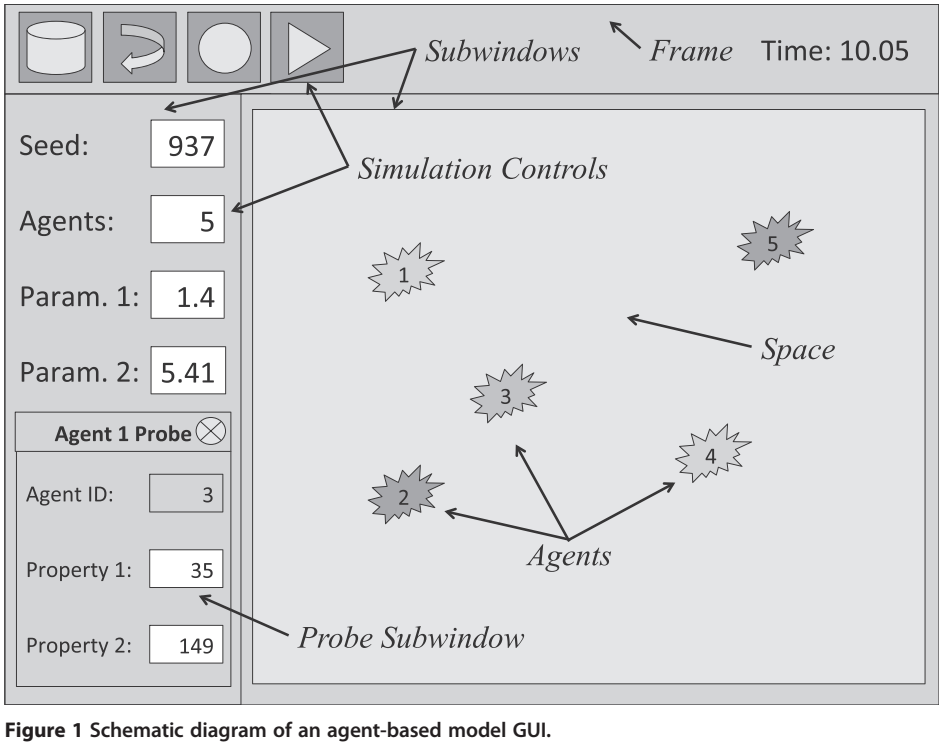

Figure 1 Schematic diagram of an agent-based model GUI.

correspondence to real computer systems than do other typical options, such as Turing machines or the λ-calculus.

A RASP is an enhanced random access machine (RAM) (Cook and Reckhow 1973). Several variations exist (Elgot and Robinson 1964; Cook and Reckhow 1973). This paper uses the formulation by Cook and Reckhow (1973).

A RAM consists of an infinite set of memory registers that are used by a finite program. To better match the real implementations in this paper, a large but finite number of registers will be used instead of an infinite number. Each register can hold an integer. Register i is denoted X_i for a nonnegative integer.

Memory will be managed using a doubly linked free list with a fixed block size combined with a free boundary reference. The free boundary reference points to the first free register beyond which all registers are unallocated. To avoid external memory fragmentation, the block size will be set to be the maximum of any the dynamically allocatable uses of memory. Memory allocation will use the first block in the free list, if one is available. If the list is empty, then a block will be created by using and then incrementing the free boundary reference. Memory deallocation will insert the freed memory at the head of the list. In this case, the free boundary reference will remain the same.

By default, a RASP only processes integers. Finite computable real numbers will also be included in our modified machine. The operations for computable real numbers will work in the same fundamental way as the corresponding integer operations defined later in this section.

Cook and Reckhow (1973) scale the sizes of their registers independently to match the values held in storage. Modern computers rarely allow such variation. Therefore, we will set the register size for each RASP run to be a RASP-wide constant value, w. Naturally, w defines the maximum value of any integer or computable real number that can be directly processed in the given program run. w will vary between runs to reflect the widths of the numbers used in the problems being solved. The size of w is discussed further, once agents are introduced.

A RAM has access to a sequential input tape and a sequential output tape. The RASP enhancements add an accumulator (AC) that contains an integer; an instruction counter (IC) that contains the address of the next instruction; the ability to run programs by sequentially reading, decoding, and executing the values in the registers pointed to by IC; and an ALGOL-like programming language that can be translated into instructions for the underlying RAM. The RASP instructions are shown in Table 1.

For large w and constants c_k, all of the operations shown in Table 1 take $O(c_k w) = O(w)$ time. In particular, processing integers and computable real numbers requires $O(w)$ time and space. Operations that allocate new registers take $O(0)$ time and $O(w)$ space.

Cook and Reckhow (1973) show that a RASP has higher computing power than a standard multi-tape Turing machine—but only by the square root of the time. The RASP architecture was chosen for this paper because it is amenable to complexity analysis while still retaining an architecture closely resembling commonly used computers.

In the next sections, the abstract machine's program and memory layout is presented. The outline intentionally uses standard data structures and algorithms to insure that the expected code is straightforward.

Table 1 RASP instructions adapted from Cook and Reckhow's (1973) Table 2

Instruction	Interpretation	Code	Time	New space
`LOD c`	Load constant **c** into **AC**	1	2**w**	0
	Increment **IC**			
`ADD` X_i	Add the value in register X_j to **AC**	2	4**w**	0
	Increment **IC**			
`SUB` X_i	Subtract the value in register X_j from **AC**	3	4**w**	0
	Increment **IC**			
`STO` X_i	Store the value in **AC** to register X_j	4	3**w**	**w**, if X_j was not allocated else 0
	Increment **IC**			
`BPA c`	If **AC** > 0, then set **IC** to **c**	5	3**w**	0
	If **AC** ≤ 0, increment **IC**			
`RD` X_i	Load the next input tape value into X_j	6	3**w**	**w**, if X_j was not
	Increment **IC**			allocated else 0
`PRI` X_i	Output the value in X_j to the output tape	7	3**w**	0
	Increment **IC**			
`HLT`	Halt execution	-∞ o 0 and 8 to ∞-	2**w**	0

Agents

Agents will be represented using a set of stored programs and a set of registers for data values, as shown in Table 2. h is the number of different types of agents in a given model. n_i is the maximum number of active agents for agent type i in a given model run. The n_i values count agents that execute at least one behavior, have attributes that are accessed at least once for each agent, or are included in at least one calculation. Completely inactive agents are not included in the n_i counts. We then have the total number of agents, N, as the sum of n_i from 1 to h. We set $w \geq \lceil \log_2(N) \rceil$ to insure that agent counts can be properly calculated in all cases. To minimize time and space requirements, we then have $O(w) = \max\ (O(\lceil \log_2(N) \rceil), Z)$, for Z the widest numeric value to be directly processed by a given model.

Agent definitions are read from input files. The input files have either individual agent descriptions or distributions. If the agents are stored individually, then there will be at least one input per agent. If distributions are used, then the needed agents are created using random draws from the given distributions.

Table 2 Agent storage layout

Register	Contents
`I`	The first of **j** registers needed to store the agent attributes.
`i + j – 1`	The last of **j** registers needed to store the agent attributes.
`i + j`	The head of a linked list used to store dynamically generated agent behaviors, if any.
`i + j + 1`	The first of **k** registers needed to store the heads of doubly linked lists used to store network links for network **1**.
`i + j + k`	The last of **k** registers needed to store the heads of doubly linked lists used to store network links for network **k**.
`i + j + k + 1`	The address of the next agent in the agent list.
`i + j + k + 2`	The address of the previous agent in the agent list.

Agents will be stored using the layout shown in Table 2. The default identifier for each agent is its lowest memory address.

Agents can have both statically defined and dynamically generated behaviors. Statically defined behaviors are included in the code for a given RASP instance. Dynamically generated behaviors are stored in an agent-specific linked list as shown in Table 2.

A doubly linked list of agents is maintained by the system. Inserting or removing agents updates this list. This list is used for activities discussed later such as time scheduling.

Basic operations to be provided by the agent-based modeling library are as follows:

- Creating an agent requires initializing the agent register storage, as shown in Table 2. This operation has time $O(r_1\lceil \log_2(N)\rceil)$ complexity and $O(r_1\lceil \log_2(N)\rceil)$ space complexity for r_1, the number of registers used by the agent. r_1 is a constant relative to N.
- Disposing of an agent requires adding the agent's space back to the free list, which takes $O(\lceil \log_2(N)\rceil)$ time.
- Changing the value of an agent attribute for a given agent j uses O(0) space and $O(r_2\lceil \log_2(N)\rceil)$ time for r_2, the required number of registers. r_2 is again a constant relative to N.
- Modifying an agent's behavior involves writing a new routine for the agent. This action has $O(r_3\lceil \log_2(N)\rceil)$ time complexity and $O(r_3\lceil \log_2(N)\rceil)$ space complexity for r_3, the required number of write operations. Statically compiled agent behaviors are stored with the main model program. Dynamically generated agent behaviors are added to an agent-specific linked list, as noted in Table 2. r_3 is yet again a constant relative to N.
- Executing an agent behavior takes time and space as determined by the contents of the corresponding routine.

The agents are accompanied by information on their surrounding environment. This information includes the statically compiled agent behaviors and other pieces of overall control information stored with the main model program. This information is in addition to a description of the interaction spaces that are present, which are discussed in a later section. The space needed to store information on the surrounding environment is always small and constant compared to that needed to store information about agents for any substantial number of agents. Therefore, environments use $O(A\lceil \log_2(N)\rceil)$ space and take $O(B\lceil \log_2(N)\rceil)$ access time for constants A and B. The need for a large amount of information on the surrounding environment implies that the agent-based model has been designed incorrectly and that agents have been blended into the environment.

Scheduler

Both time-stepped and discrete event scheduling are available. Each model will use one of these options exclusively.

The time-stepped scheduler uses the agent list and an integer time counter. This scheduler simply picks elements sequentially from the agent list and executes the next behavior. Every agent runs for each time step. Each time-stepped agent has a default 'step' routine that is called for each time step. A given agent's step routine can be empty

if no behavior is needed for a given time step. This default is taken to be the first element in the linked list of dynamically generated agent behaviors, if any. If an agent's dynamic procedures list is empty, then a call is made to the first routine in the list of statically compiled functions for the corresponding agent type. Whichever way the step routine is found, it can call other routines as needed.

The discrete scheduler uses a time counter and an event time slot list stored as a B-tree sorted by time stamp. Each event time slot list entry includes a time value and the head pointer of a singly linked list of events. Each event, in turn, contains a time stamp, the address of the routine to be invoked, a single register value that can contain an optional parameter, and a link to the next list element. Decoding the parameter is solely the responsibility of the routine to be invoked. Inserting a new event time slot into the event time slot list, in the worst case, requires $O(\lceil \log_2(N) \rceil \log v)$ time for a list with v unique time slots. Finding an event time in the worst case takes $O(\lceil \log_2(N) \rceil \log v)$. Deleting an event when it is completed in the worst case takes $O(\lceil \log_2(N) \rceil \log v)$. The B-tree will use space $O(v\lceil \log_2(N) \rceil)$. In the worst case inserting, reading, or deleting an event once the corresponding event time has been found or inserted takes $O(\lceil \log_2(N) \rceil)$ time.

A model is said to have asymptotically linearithmic[a] scheduling in time or space when the time or space performance is $O(N\lceil \log_2(N) \rceil)$. Time step scheduling is always asymptotically linearithmic in both time and space. Discrete scheduling is asymptotically linearithmic in time and space when the number of pending unique discrete event time slots, referred to as v previously, is constant with respect to N.

Regardless of the kind of scheduling that is employed, the number of time steps or unique discrete event times is usually independent of N. In other words, models are rarely run for more time steps or unique discrete event times because they have more agents. We thus assume that the length (i.e., time steps or slots) of each run is a constant relative to N, and therefore run length does not affect asymptotic time or space performance. We will briefly return to this assumption in a later section.

Interaction spaces

Three kinds of interaction spaces are supported. Each model can have any number of these spaces, although in practice the number of different interaction spaces is usually quite small.

Scalar fields will be stored as multidimensional arrays indexed in row major order. Each d-dimensional scalar field with maximum dimension range g will use space $O(g^d\lceil \log_2(N) \rceil)$. Each grid point holds a computable real number or an integer. An individual value can be read or written in $O(\lceil \log_2(N) \rceil)$ time.

Grids will also be stored as multidimensional arrays indexed in row major order. Each array element will store the head pointers of doubly linked lists of agent records and a counter of the length of each list. There is one head pointer and counter pair for each agent type. Each agent record holds the reference of the associated agent and the agent's floating point coordinates. Agents will also keep a reference to their own records. The doubly linked list can be of any length between zero and N. It is up to each modeler to process these lists in an appropriately efficient way. Each d-dimensional grid with width g will use space $O(g^d\lceil \log_2(N) \rceil)$. An individual value can be read or written in $O(\lceil \log_2(N) \rceil)$ time.

Networks will be stored using a doubly linked list of edges for each agent. The head references for each agent are shown in Table 2. Each linked list element contains the identifier of the linked agent, a computable real number weight, a reference to the next element, if any, and a reference to the previous element, if any. The use of weights by agent behaviors is optional. Undirected networks contain linked lists with paired references (i.e., if agent A's list includes agent B, then B's list will also include A). Directed networks allow lists with asymmetric references. Each network will use $O(bN\lceil \log_2(N) \rceil)$ space for b the maximum number of links per agent. An agent can insert, read, or delete an entry in their list in $O(b\lceil \log_2(N) \rceil)$ time.

Random number stream

A pseudorandom number generation algorithm with $O(\lceil \log_2(N) \rceil)$ asymptotic time and space performance will be used to produce a uniformly distributed random number stream. The Mersenne Twister (Matsumoto and Nishimura 1998) is an example, at least until the number of random draws approaches the algorithm's period of 2^{19937}-1. Non-uniform distributions can be provided by applying appropriate constant time functions to the core uniform random number stream. Also, the random seed used for stochastic execution is an input parameter.

Logging

The log stores values to the output tape using the PRI instruction. The log will be unstructured. This will allow users to determine the format of the logs individually for each model. The PRI instruction uses $O(\lceil \log_2(N) \rceil)$ time and O(0) space for each value to be logged.

User interfaces

The abstract machine will not normally use a GUI. As previously discussed, the machine will typically execute in a headless batch mode, since we are analyzing asymptotic (i.e., large-scale) performance. Nonetheless, the performance impact of a GUI will be briefly analyzed.

The abstract machine model can accommodate a GUI by including an optional memory region on startup that represents the graphics buffer. The machine will then include routines to draw agents into the memory locations representing the graphics buffer.

Drawing an agent-based model GUI usually involves rendering the frame and subwindows shown in Figure 1. The complexity of the overall frame and simulation control subwindows are independent of the number of agents. Thus, these components run in constant time and use constant space relative to the number of agents.

Users normally invoke probes manually rather than programmatically. As such, there is a fixed upper bound on the number of agents that can realistically be shown with probes in a GUI. Therefore, probes also run in constant time and use constant space relative to N.

Drawing the interaction spaces usually involves rendering many, if not all, of the agents individually. Generally, it will take $O(\lceil \log_2(N) \rceil)$ time to draw each view of the agents. Optimizations that reduce the number of agents considered for drawing may

improve the time complexity. For example, simply not showing interaction spaces reduces the complexity of this task to O(0).

Drawing can be performed on an individual basis with intermediate results accumulating in the graphics buffer (i.e., graphics memory). Graphics buffers have a fixed size that is related to the screen resolution and is independent of the number of agents. Thus, drawing takes constant storage space.

The remainder of this paper will focus on batch mode model runs. The analysis that follows can be augmented using the results in this section, as needed.

Parallel execution

There are multiple ways to use parallel computing to improve simulation performance Perumulla (2006), Parker and Epstein (2011) and Collier and North (2012) present examples. Two major paths are parallel parameter sweeps and distributed execution of individual simulations. Obviously, these approaches can be combined. Both paths will be considered in the next sections.

Parallel parameter sweeps

Simultaneously executing each run of a parameter sweep is sometimes called an "embarrassingly parallel" approach because of the relative simplicity of its implementation compared to more sophisticated alternatives (Foster 1995). The speedup factor is just the number of simultaneous runs discounted for any overhead required for job control. The overhead for this type of implementation is generally quite low, as implied by the "embarrassingly parallel" moniker. The major limiting factor is the number of processors, p, available for executing model runs. For even the largest computers, p is constant relative to N. Therefore such runs do not affect the asymptotic time requirements. Similarly, the memory requirements for a parallel parameter sweep will be the size of a model run times the number of simultaneous runs, again with additions for job control requirements and with discounts for possible shared memory. These factors, combined with the constant number of processing nodes, imply that parallel parameter sweeps do not affect the asymptotic space requirements.

Distributed model runs

Executing a single model using many processing nodes offers the potential to speed up the delivery of individual results. Here we will assume an ideal implementation. In this case, the asymptotic space requirements remain unchanged, given that the same number of agents needs to be tracked as with a sequential model. The best speedup for asymptotic time is 1/p for p processors because no computing work can be ignored. Unfortunately, as previously discussed, p is a constant relative to the number of agents, so there is no asymptotic time speedup.

Modeling studies

Agent-based modeling studies often require multiple model runs to cover different scenarios and to account for stochastic variation. The time and space complexity results for individual runs can be used to estimate the requirements for each needed type of run and then multiplied by the corresponding number of runs and summed, as appropriate.

In many situations, even though the number of model runs needed for a given study is a factor of many input variables, it is independent of the number of agents. In these cases, the asymptotic time and space requirements, of course, are the same as for an individual model run.

The abstract machine reads the first input tape value to determine the number of input sets that are present. If more that one input set is present then they are simply concatenated in the desired order of execution.

The abstract machine has constant time routines to reset itself and start the new model runs, as needed. The output from each run is simply appended to the output tape in order of execution. It is the responsibility of modelers to store appropriate delimiters or use other encodings to differentiate output run results.

The abstract machine has an optional provision for executing a routine to aggregate a set of runs after they have completed. The performance of this routine is included in the overall performance of the data aggregation system. In general, if the number of model runs is constant with respect to N and the model-level data aggregation is linearithmic or better in asymptotic time and space performance then the parameter sweep-level data aggregation routine will also be linearithmic or better in performance.

Results and discussion

Best cases for any model

This section presents results that apply to all models. Considering memory usage, there must be an assignment operation for every register that is used beyond the size of the basic machine and model program, both of which are constant with respect to N. Therefore, for maximum model memory space usage S_{max}, we have:

(1) For any model, $O(T_{total}) \geq O(S_{max})$.

Considering output logs, there must be a PRI instruction for each output value. For total time T_{total} and total output log space usage S_{log} we therefore have:

(2) For any model, $O(T_{total}) \geq O(S_{log})$.

Best cases for asymptotically thorough and incompressible models

Two important definitions are presented in this section. These definitions are then used to analyze the best cases for time and space.

Asymptotically thorough models

A model with N agents is said to be *asymptotically thorough* when individual values from O(N) agents are needed from at least two different representative time periods to correctly calculate the logged results. Representative means that the span between logged time periods includes typical agent behaviors. Logging a value from each agent for two or more normal time steps is generally sufficient to meet this standard.

Asymptotically incompressible models

A model with N agents is said to be *asymptotically incompressible* when a minimum of O(N) space is needed to store the agents and recalculating an attribute for an agent takes at least as much space as storing it between uses. In other words, the best compression ratio achievable for agent storage is constant relative to N for the general case for a given model. Agent attributes that can take on a wide range of values are generally sufficient to meet this standard.

Analysis

We will now consider the best cases for time and space performance for asymptotically incompressible and asymptotically thorough models.

Time

For N decision makers, any asymptotically thorough model must access the attributes for at least O(N) decision makers for a minimum of two different time periods. Thus, the output aggregation must use at least O(N) operations, each of which takes $O(\lceil \log_2(N) \rceil)$ time. Then we have the following:

(3) The minimum asymptotic time needed for asymptotically thorough models is in $\Omega\ O(N\lceil \log_2(N) \rceil)$ for variable word size computers.

Space

Asymptotically thorough models must log individual values from O(N) agents from at least two different representative time periods to correctly calculate the logged results. Any asymptotically incompressible model must either store the relevant decision maker attributes between the first and second time periods or use at least as much space recreating them for the second logging period. Either way, this requires $O(N\lceil \log_2(N) \rceil)$ space. Therefore, we have the following:

(4) The minimum asymptotic space needed for asymptotically thorough and incompressible model is in $\Omega\ (N\lceil \log_2(N) \rceil)$ for variable word size computers.

Time-stepped, boundedly rational modular imperative agent-based model worst cases

In this section, we will consider the worst cases for time and space for asymptotically thorough and incompressible batch-mode modular imperative agent-based models with large numbers of individual, boundedly rational, decision makers; input parameter ranges and counts that do not increase with the number of decision makers; asymptotically linearithmic scheduling; and output aggregation that is linearithmic or faster in time and space. Obviously, it is assumed that the agent-based model in question is suitable for the problem, correctly written, and efficiently implemented relative to the constraints and requirements of the question to be answered.

Time

The N agents in a model with, at most, s unique event times (i.e., times steps or unique discrete event times) and d operations in the agent routine with the greatest time complexity requires, at most, $O(Nsd\lceil \log_2(N) \rceil)$ time. s does not increase with N. Thus, s is a constant with respect to N. Bounded rationality means in part that the amount of computation that can be completed by any one agent in a single time step is strictly limited. This

limitation, in turn, means that d must be a constant relative to N because N is unbounded. All of the agent's operations are either programmed using the instruction set given in Table 1 or are interaction space access calls. Thus, the agents themselves require, at most, $O(N\lceil \log_2(N)\rceil)$ time plus access time for any interaction spaces that might be used.

As discussed above, environments use $O(\lceil \log_2(N)\rceil)$ time. There is only one environment. Therefore, the total time N agents spend accessing information in the environment is, at most, $O(N\lceil \log_2(N)\rceil)$.

Scalar fields can be accessed in $O(\lceil \log_2(N)\rceil)$ time. The number of scalar field accesses in each agent behavior must be bounded due to bounded rationality. Therefore, the total time N agents spend accessing scalar fields is, at most, $O(C_1N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$, where C_1 is the maximum number of accesses in an agent behavior.

Grids are stored as simple multidimensional arrays. The access time is just the time needed for array indexing, namely, $O(\lceil \log_2(N)\rceil)$ per access. The number of linked list checks and overall grid accesses in each agent behavior must be bounded due to bounded rationality. Therefore, the total time N agents spend accessing grids is, at most, $O(C_2N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$, where C_2 is the maximum number of grid accesses in an agent behavior.

Each network has $O(b\lceil \log_2(N)\rceil)$ access time for b, the maximum number of links per agent. Bounded rationality implies that the number of network links that can be remembered by any one agent is strictly limited. This constraint, in turn, means that b must be a constant relative to N, given that N is unbounded. The number of accesses must also be bounded due to bounded rationality. Therefore, the total time N agents spend accessing networks is, at most, $O(C_3bN\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$, where C_3 is the maximum number of network accesses in an agent behavior.

Output aggregation uses linearithmic or less time. The outputs, therefore, require, at most, $O(N\lceil \log_2(N)\rceil)$ time in addition to that used by the rest of the model.

We have the following for the total execution time, T_{total}:

$$T_{total} \leq T_{agents} + T_{environment} + T_{fields} + T_{grids} + T_{networks} + T_{outputs}$$

The count of each type of interaction space will be a constant relative to N as previously discussed. Substituting in the findings thus far produces the following:

$$T_{total} \leq O(N\log_2(N)) + O(N\log_2(N)) + O(N\log_2(N)) + O(N\log_2(N)) + O(N\log_2(N)) = O(N\log_2(N))$$

Therefore, $T_{total} \in O(N\log_2(N))$.

The input parameter ranges and counts do not increase with the number of decision makers. This constraint means that the number of model runs, M, needed to solve a particular problem instance is, at worst, constant relative to N. Thus, if one run requires, at most, $O(N\lceil \log_2(N)\rceil)$ time, then a proper design of experiments for a study to answer a specific question requires $O(MN\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ time. Thus:

(5) The maximum asymptotic time for studies done with asymptotically thorough and incompressible batch mode modular imperative agent-based models with large numbers of individual, boundedly rational, decision makers; input parameter ranges and counts that do not increase with the number of decision makers; asymptotically

linearithmic scheduling; and output aggregation that is linearithmic or better in time and space is $\Omega(N\lceil \log_2(N)\rceil)$ for variable word size computers.

Space

Modular imperative agent-based models read all of their agent inputs, create the corresponding agents, and then store them. The combined requirements of asymptotic thoroughness and incompressibility prevent agent data from being recreated for each time step to save space. Thus, $O(N\lceil \log_2(N)\rceil)$ space is required.

Environments use $O(C_4\lceil \log_2(N)\rceil)$ space for constant C_4. There is one environment. Therefore, environments use, at most, $O(\lceil \log_2(N)\rceil)$ space.

Each scalar field requires $O(g^d\lceil \log_2(N)\rceil)$ space. d will be a constant relative to N, since the number of space dimensions normally does not depend on the number of agents. The growth rate for the maximum dimension range, g, is tied to the agent space density, y, given by $y = N/g^d$, as covered by these two cases:

1. g grows asymptotically faster than $N^{1/d}$. In this case, the agent density will drop and eventually either the agents will cluster or most agents will be isolated from one another. In either case, the model's use of the space becomes trivial. The space can then be replaced with either a set of smaller spaces when most of the agents cluster or with a network when most of the agents are isolated from one another. Thus, this case is not allowed.
2. g grows at the same rate or asymptotically slower than $N^{1/d}$. The agent density is constant or rising so the agent interactions can continue to be mediated via the field. The overall space usage for the grid is then $O(g^d\lceil \log_2(N)\rceil) \le O(((N/y)^{1/d})^d \lceil \log_2(N)\rceil) = O(((N/y)\lceil \log_2(N)\rceil) \le O(N\lceil \log_2(N)\rceil)$

The number of scalar fields must be bounded due to bounded rationality. Therefore, scalar fields use $O(C_5N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ or less space, for C_5 scalar fields.

Each grid requires $O(g^d\lceil \log_2(N)\rceil)$ space. d will be a constant relative to N. The doubly linked grid point membership lists can individually vary in size, but in total they take up $O(N\lceil \log_2(N)\rceil)$ space for N agents. The growth rate for g is tied to the agent space density, y, given by $y = N/g^d$, as covered by these two cases:

1. g grows asymptotically faster than $N^{1/d}$. In this case, the agent density will drop and eventually either the agents cluster or most agents will be isolated from one another. In either case, the model's use of the space becomes trivial. The space can then be replaced with either a set of smaller spaces when most of the agents cluster or with a network when most of the agents are isolated from one another. Thus, this case is not allowed.
2. g grows at the same rate or asymptotically slower than $N^{1/d}$. The agent density is constant or rising so the agent interactions can continue to be mediated via the field. The overall space usage for the grid is then $O(g^d\lceil \log_2(N)\rceil) \le O(((N/y)^{1/d})^d \lceil \log_2(N)\rceil) = O(((N/y)\lceil \log_2(N)\rceil) \le O(N\lceil \log_2(N)\rceil)$

The number of grids must be bounded due to bounded rationality. Therefore, grids use, at most, $O(C_6N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ or less space, for C_6 grids. Of course,

agents must be selective in how they access the grid point membership lists to maintain bounded rationality.

Each network requires $O(bN\lceil \log_2(N)\rceil)$ space. b has been previously shown to be a constant relative to N. Thus, each network requires $O(N\lceil \log_2(N)\rceil)$ space. As with the other kinds of spaces, the number of networks must be bounded due to bounded rationality. Therefore, networks use, at most, $O(C_7N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ space, for C_7 networks.

Output aggregation uses linearithmic or less space. The outputs, therefore, require at most $O(N\lceil \log_2(N)\rceil)$ space in addition to that used by the rest of the model.

The input parameter ranges and counts do not increase with the number of decision makers. This means that the number of model runs, M, needed to solve a particular problem instance is at worst constant relative to N. Thus, if one run requires at most $O(N\lceil \log_2(N)\rceil)$ space, then a proper design of experiments for a study to answer a specific question requires $O(MN\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ space. Thus:

(6) The maximum asymptotic space for studies done with asymptotically thorough and incompressible batch mode modular imperative agent-based models with large numbers of individual, boundedly rational decision makers; input parameter ranges and counts that do not increase with the number of decision makers; asymptotically linearithmic scheduling; and output aggregation that is linearithmic or faster in time and space is $\Omega\ O(N\lceil \log_2(N)\rceil)$ for variable word size computers.

Computational optimality

Next, we consider the conditions under which modular imperative agent-based modeling studies may be considered to have optimal asymptotic time or space performance. As previously stated, (optimal) means that no other modeling technique can solve the same problem as an agent-based model can using fewer resources — in this case, time and space. The strategy is straightforward because both the lower bounds on all modeling techniques and the upper bounds on a common class of agent-based models have been proven. Using this background, we will now prove the following:

(7) Studies with modular imperative agent-based models are computationally optimal in asymptotic time and space performance if the models are asymptotically incompressible; asymptotically thorough; have large numbers of individual, boundedly rational, decision makers; have input parameter ranges and counts that do not increase with the number of decision makers; use batch mode operation; have asymptotically linearithmic scheduling; and have output aggregation that is linearithmic or faster in time and space for variable word size computers.

(3) and (4) show that the best case time and space bounds for any asymptotically incompressible and thorough model are both in $\Omega\ (N\lceil \log_2(N)\rceil)$. We also showed that individual, boundedly rational decision makers in an asymptotically incompressible and thorough agent-based model with individual inputs requires $O(N\lceil \log_2(N)\rceil)$ time and space, plus the time and space needed for accessing the interaction spaces. We additionally showed that the agent-based modeling interaction spaces require, at most,

$O(N\lceil \log_2(N)\rceil)$ time and space for each space for N boundedly rational agents. This formulation yields $O(N\lceil \log_2(N)\rceil) + O(C_8N\lceil \log_2(N)\rceil) = O(N\lceil \log_2(N)\rceil)$ total asymptotic time and space performance for C_8 spaces. Finally, we showed that modeling studies and parallel runs do not change the asymptotic time and space complexity of agent-based models. Therefore, (7) is proven.

Given (2) we also have:

(8) Modular imperative agent-based models are computationally optimal in asymptotic time performance if $O(T_{total}) = O(S_{log})$.

Computational power of modular imperative agent-based models

In this section, we use the formalism to investigate other properties of agent-based models. Here we focus on characterizing the computational generality of modular imperative agent-based modeling.

Modular imperative agent-based models are computationally complete

Any RASP algorithm can be stored as the sole behavior for a single agent. Executing the behavior can be the sole scheduled event for the model. The inputs for the algorithm can be stored as the input data for the solitary agent in the model. All of these mappings take constant time and space relative to the problem size. Memory allocation time and space are at worst a constant multiple of register assignment costs. The algorithm can thus use a standard RASP approach and produce results and performance metrics asymptotically identical to a default RASP. RASPs have been proven to be a general model of computation, subject to the correctness of the Church-Turing thesis. The RASP formulation of agent-based models thus has the same computing power and performance as a general RASP. Therefore:

(9) Modular imperative agent-based models are computationally complete, subject to the correctness of the Church-Turing thesis.

Hartmanis (1970) and Luginbuhl (1970) discuss the computational complexity of RASPs in detail. Naturally, agent-based models have the same computational complexity as general RASPs.

Modular imperative agent-based models are universal models

Result (9) says in part that if the Church-Turing thesis is correct, then any computable model output can be computed with modular imperative agent-based modeling. Therefore:

(10) Any computable model can be expressed as a modular imperative agent-based model.

For example, Macal (2010) presents an algorithm to convert any system dynamics model to a modular imperative agent-based model. Of course, result (10) does not mean that modular imperative agent-based modeling should be automatically used to solve all modeling problems. The practical circumstances surrounding a given modeling problem need to be taken into account as well.

Modular imperative agent-based model complexity classes: ABM, ABMSPACE, and ABMTIME

Let ABMTIME be the set of all modular imperative agent-based models that have optimal time performance. Let ABMSPACE be the set of all modular imperative agent-based models that have optimal space usage. Then let ABM be the set of all modular imperative agent-based models that have optimal time and space performance. Naturally:

(11) ABM = ABMTIME ∩ ABMSPACE

ABMTIME

What can we say about ABMTIME? We can construct a set of models. Let g(N) be a valid complexity class that is linearithmic or larger. Then let $g'(N) = (N^{-1})g(N)$. We have $g'(N) \geq 1$ since g(N) is linearithmic or larger. Define a modular imperative agent-based model with N agents and no interaction spaces. Assign each agent a random number 'tag' so that the set of agents is asymptotically incompressible. Program each agent to have a constant time interaction with $\lceil g'(N) \rceil$ other randomly selected agents for each of a fixed number of time steps. The output is a log entry of each interaction so the model is asymptotically thorough. This model is optimal according to result (7). It has $O(g'(N)) = O(N(N^{-1})g(N)) = O(g(N))$ asymptotic time performance. Therefore:

(12) ABMTIME includes modular imperative agent-based models of all complexity classes linearithmic or larger.

ABMSPACE

Similarly, what can we say about ABMSPACE? We can again construct a set of models. Begin with the model used for the proof of (12). Change the agents so they store the tag for each interaction from the previous time. Also, have the agents compute a new tag as the sum of the tags from the last interaction. This is $O(g'(N)) = O(N(N^{-1})g(N)) = O(g(N))$ asymptotic incompressible space usage As before, this model is optimal according to result (7). Thus:

(13) ABMSPACE includes modular imperative agent-based models of all complexity classes linearithmic or larger.

Toward real computers

In the proceeding sections, we have assumed an abstract model of computation that scales with the size of N. In this section, we will briefly relax this assumption to better approximate real computers. Even in this section, the relaxation will only be partial, since we want the formalism to accommodate the full range of typical computers. The relaxation here will occur along two axes, namely word size and run length.

Fixed word size

Real computers generally have fixed word sizes. In this case, the $\lceil \log_2(N) \rceil$ term we previously used to account for increasing word size for increasing N becomes 1. The results in the proceeding sections can then be simplified by replacing the $\lceil \log_2(N) \rceil$

term with 1 and replacing the linearithmic requirements with linear requirements. In particular, result (7) becomes the following:

(14) Studies with modular imperative agent-based models are computationally optimal in asymptotic time and space performance if the models are asymptotically incompressible; asymptotically thorough; have large numbers of individual, boundedly rational, decision makers; have input parameter ranges and counts that do not increase with the number of decision makers; use batch mode operation; have asymptotically linear scheduling; and have output aggregation that is linear or faster in time and space on fixed word size computers.

Substantial run length

The performance bounds in the proceeding sections included the observation that the number of time steps or unique discrete event times is generally independent of N. Therefore, these factors do not affect asymptotic time or space performance. For real computers, a substantial constant may still affect performance. In this case, the time performance results should be multiplied by s, the number of time steps or slots. The space performance results remain the same for boundedly rational agents with linear output aggregation on fixed word size computers. Here bounded rationality limits the agent's ability to accumulate memory, and linear output aggregation storage constrains output aggregation space.

Examples

Returning to the cellular-level human tissue growth example discussed in the introduction, if N is the number of cells, each cell's behavior is boundedly rational, cell density is finite, and the cells exist in three dimensional grid then the model will be O(N) on fixed word size computers.

In the following sections we analyze the asymptotic time and space performance of three example modular imperative agent-based models. The examples were chosen from the set of published agent-based models, so as to cover a range of different designs and a spectrum of varying uses. These examples show how to apply both the formalism and the optimality conditions to several different kinds of real agent-based models. These examples also demonstrate how the formalism is useful for predicting the execution time and space requirements for representations of common CASs.

A simple synthetic model

The Lotka-Volterra (Lotka 1925; Volterra 1926) predator–prey model is a classic differential equations model that is commonly implemented using system dynamics. Here we will consider a three species 'predator–prey-plant' variant. In this model, there are predators that eat prey and prey that eat plants. The animals (i.e., the predators and prey) have an energy level and die if their energy goes to zero. Plants regrow at a specified rate. N is the sum of the three species' populations. By convention, the output is a population count time series for each tracked species or compartment. The input parameter ranges and counts do not increase with N.

Theoretical analysis

The modular imperative agent-based predator–prey-plant implementation has one agent type for each species and is time-stepped. Since it is time-stepped, it has

asymptotically linear scheduling. A selected Repast Simphony implementation is used for reference.

For the Repast Simphony implementation, animals reproduce probabilistically using random draws checked against reproduction rate thresholds. The animal agents move on a two dimensional toroidal grid, with plant agents at each point. During each time step, the animals move to a random point near their current location and then eat if the appropriate food is available. This is a simple boundedly rational $O(a\lceil \log_2(a)\rceil)$ time and space behavior for a animals on variable word size computers.

Plants can be living or dead. Live plants grow at a constant rate. Areas with dead plants lie fallow for a period of time and then life returns. Plants die due to excessive consumption by prey. This is a simple boundedly rational $O(b\lceil \log_2(b)\rceil)$ time and space behavior for b plants.

a and b can both be large. With $N = a + b$, we have N large, as well.

The three species each maintain individual energy levels that can take on any double precision value. As such, they are asymptotically incompressible.

Combining results (5) and (6) with the minimum requirements that were just discussed implies that this model is $O(N\lceil \log_2(N)\rceil)$ in time and space. This version of the agent-based model does not qualify for time or space optimality under result (14) because the output requires only compressible information from each agent (i.e., the population count) rather than specific attributes. In other words, the model's output aggregation is not asymptotically thorough. Is there a more efficient implementation?

The system dynamics predator–prey-plant implementation has one stock for each species and a set of flows between stocks. On variable word size computers, this model has $O(C_9\lceil \log_2(N)\rceil) = O(\lceil \log_2(N)\rceil)$ asymptotic time and space performance for constant C_9 measuring the time and space usage for the fixed stocks and flows. Obviously, this model has better time and space performance than the previous agent-based version.

Consider a variation of the model used to study predator and prey migration patterns. Now, the model must output the grid locations of each animal for every time step. This agent-based model is asymptotically thorough. It remains $O(N\lceil \log_2(N)\rceil)$ in time and space but it is now optimal under result (14).

What about the $O(\lceil \log_2(N)\rceil)$ system dynamics implementation? It cannot generate the needed outputs. A scalar field approach is also insufficient, since this only shows net flows, not long distance migrations. Modifications that allow the system dynamics model to track individual animal coordinates and energy levels results in a model with $O(N^2\lceil \log_2(N)\rceil)$ terms in each of the animal coordinate and energy level equations. The result is a model that requires $O(N^2\lceil \log_2(N)\rceil)$ space and $O(N^2\lceil \log_2(N)\rceil)$ evaluation time on variable word size computers. Furthermore, the model has awkward constraints on the range of model execution, since the formulation limits the maximum number of animals allowed in a given run. Wilson (1998) further explores the relationship between agent-based modeling and system dynamics for the two species variant of this model.

Experimental results

Results for 100 runs of the Repast Simphony predator–prey-plant model for varying grid sizes and 500 time steps are shown in Figures 2 and 3. The number of time steps

is held constant with respect to N. The runs in Figures 2 and 3 were completed on a 1.8 GHz Intel Core i7 MacBook Air with 4 GB of memory.

For fixed word-size computers, we have $O(N\lceil \log_2(N) \rceil) = O(N)$, which is a linear relationship. As predicted, the test measurements of the predator–prey-plant model execution time are collectively linear versus N with an adjusted R^2 of 0.7794 and a p-value less than 2.200×10^{-16}. The space measurements are also collectively linear versus N with an adjusted R^2 of 0.5687 and a p-value less than 2.200×10^{-16}.

A research model

Strains of the bacterium community-associated methicillin-resistant *Staphylococcus aureus* (CA-MRSA), are responsible for potentially life-threatening infections of the skin, soft tissue, blood, bone, and other human tissues. CA-MRSA strains are resistant to standard antibiotics related to penicillins and have a high prevalence in the general community, as well as in healthcare facilities. Macal et al. (2012) have developed two fine-grained, modular imperative agent-based models of CA-MRSA for the Chicago metropolitan area. These hourly time-stepped models feature people as agents. Both models share a common design. We will consider the large-scale Repast HPC version here due to its ability to scale across a wide range of agent population sizes. We will use this ability to compare the theoretical predictions of our time and space performance model to measured results over several orders of magnitude for N. We will also compare this simulation to system dynamics epidemic simulations.

Theoretical analysis

The Repast HPC CA-MRSA modular imperative model has N person agents. The number of agents is constant throughout each simulation run. The number of agents is generally large because the model is used to study major parts or all of the Chicago area. The agents have activity schedules and a list of locations at which to act. The activities are coded by hour, location, and activity type. The agent activities can be interrupted when problems such as hospitalization or incarceration occur.

The model uses time step scheduling with a step size of one hour. Since it is time stepped, it has asymptotically linearithmic scheduling.

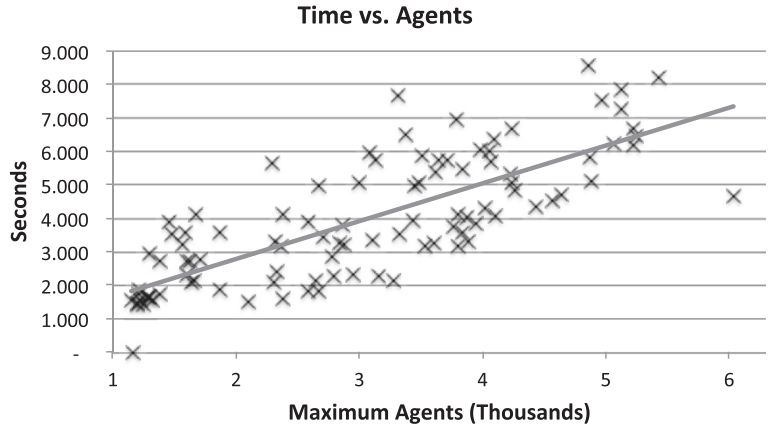

Figure 2 Predator–prey-plant model time versus N.

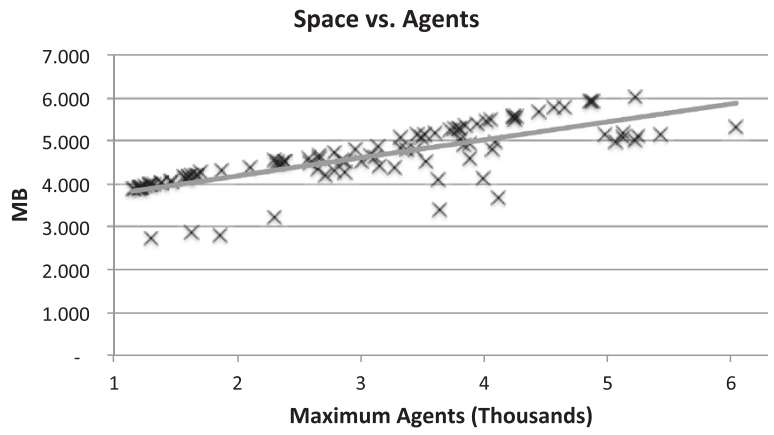

Figure 3 Predator–prey-plant model space versus N.

The location and activity type for each agent for each simulated hour contribute to the determination of the risk of disease transmission and the likelihood of disease progression. The disease states of the people in a location also contribute to the risks and likelihoods. The locations include residences such as homes or group quarters, workplaces, schools, gyms, hospitals, and jails.

The model tracks three disease states for each agent, namely uncolonized, colonized, and infected. Uncolonized people do not have the CA-MRSA. Colonized people have the disease but do not show symptoms. Infected people have symptoms and are believed to be more likely to spread the disease than are colonized people. Infected people may either treat themselves or seek medical care.

Individual behaviors involve selecting new activities from a personal list that is short and constant compared to N, moving to the selected activity location, and then interacting with the other agents at that place. The number of agents at each place is always small with respect to N. The individual, boundedly rational agent behaviors are thus O ($\lceil \log_2(N) \rceil$) for each agent.

The people in the model have a disease state that can take on any one of three values for each individual. As such, they are asymptotically incompressible.

The model produces two kinds of outputs. The first kind includes an activity entry for each agent for each time step. This is asymptotically thorough. The second kind consists of a summary line for each time step. There is a fixed number of each kind of output. Both types of outputs use straightforward data collection that is linearithmic or less time and space.

The input parameter ranges and counts do not increase with the number of people in the simulation, since the parameters provide constants for the previously described bounded rational behavior. Therefore, modeling studies do not increase the asymptotic time or space complexity. This model implementation is thus $O(N\lceil \log_2(N) \rceil)$ in time and space. Its optimality depends on which outputs are needed to answer the modeling question at issue. What about alternative techniques?

Epidemics have long been studied using system dynamics modeling (Kermack and McKendrick 1927). Compartments are generally used to track the populations of interest. System dynamics epidemic models usually have a fixed number of stocks, often one for each compartment, and a fixed set of flows between stocks. These models have

$O(C_{10}\lceil \log_2(N)\rceil) = O(\lceil \log_2(N)\rceil)$ asymptotic time and space performance for constant C_{10} measuring the time and space usage for the fixed stocks and flows. Obviously, these models have better time and space performance than the previously discussed agent-based version as long as the number of compartments is constant relative to N and limited compartments are sufficient to answer the question at hand.

Consider the use of the model to study disease contagion networks. The needed model output will be the data needed to create a disease spread dendrogram showing who contracted the tracked disease from whom. Now, the model must output the identities of the interacting parties (e.g., agents) every time the disease spreads from person to person. The first model output previously mentioned, namely hourly activity information, meets this requirement. This agent-based model is asymptotically thorough. It remains $O(N\lceil \log_2(N)\rceil)$ in time and space, but it is now optimal under result (14).

What about the $O(\lceil \log_2(N)\rceil)$ system dynamics implementation? As with the previous predator–prey-plant model, it cannot generate the needed outputs. Modifications that allow the system dynamics model to track individual disease states and contacts result in a model with $O(N^2\lceil \log_2(N)\rceil)$ terms in each person's state equation. The result is a model that requires $O(N^2\lceil \log_2(N)\rceil)$ space and $O(N^2\lceil \log_2(N)\rceil)$ evaluation time. Furthermore, the formulation fixes the maximum number of people allowed in a given run. Rahmandad and Sterman (2008) further explore the relationship between agent-based and system dynamics disease dispersion models.

Experimental results

Results for runs of the CA-MRSA model for 10 years and five geographic regions are shown in Figures 4 and 5. The regions in order of increasing population are a single ZIP code, three ZIP codes, the South Side of Chicago, and Chicago itself. Figures 4 and 5 show 32 runs for each region. The number of time steps is held constant with respect to N.

The runs in Figures 4 and 5 were completed on the 320-node Argonne National Laboratory Fusion Linux cluster. Each Fusion computing node has two Nehalem 2.6 GHz Pentium Xeon processors with 36 GB of RAM.

For fixed word-size computers, we have $O(N\lceil \log_2(N)\rceil) = O(N)$, which is a linear relationship. As predicted, the test measurements of the CA-MRSA model execution

Figure 4 Measured MRSA model time versus N.

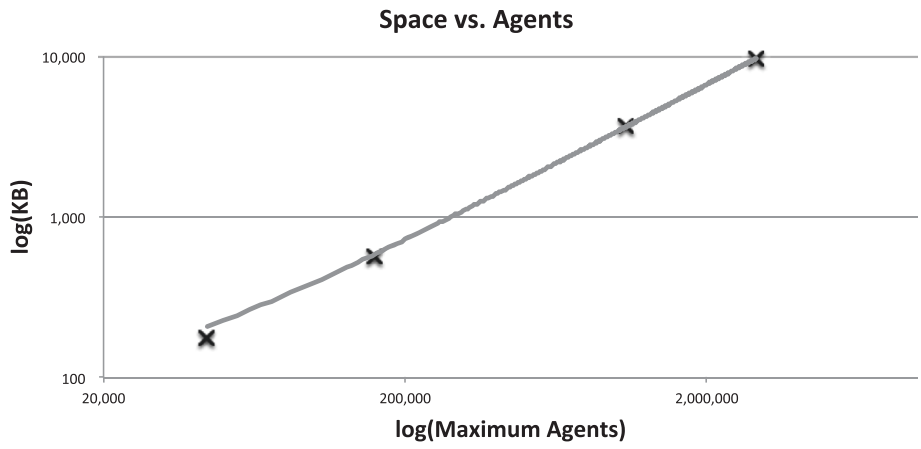

Figure 5 Measured MRSA model space versus N.

time are collectively linear versus N with an adjusted R^2 of 1.000 and a p-value less than 2.200×10^{-16}. The space measurements are also collectively linear versus N with an adjusted R^2 of 0.9999 and a p-value less than 2.200×10^{-16}.

An industrial model

The Virtual Market Learning Lab (North et al. 2010) is a large-scale modular imperative agent-based model of consumer markets co-developed by Argonne National Laboratory and Procter & Gamble (P&G). It represents the shopping behavior of consumer households and the business behavior of retailers and manufacturers in a simulated national consumer market. Argonne and P&G successfully calibrated, verified, and validated the resulting agent-based model using several independent, real-world data sets for multiple consumer product categories with more than 60 comparison tests per data set. First, Repast and then later Repast Simphony were used to implement the model. P&G has successfully applied the model to several challenging business problems, where it has directly influenced managerial decision-making and has produced substantial cost savings. The version of the model analyzed in this paper does not use social networking.

Theoretical analysis

The model uses time step scheduling with a step size of one day. Since it is time stepped, it has asymptotically linearithmic scheduling.

The agents in this model are consumer households (n_0), retail stores (n_1), retail regions (n_2), retailers (n_3), and manufacturers (n_4) (North et al. 2010). Here, N is the sum of n_i for i = 1 to 5. In principle, N scales with the number of any of the agent types. In practice, the number of consumers, n_0, is always orders of magnitude greater than the counts of the other agents. Thus, we have $N = n_0$. n_0 is generally large, so N is as well.

Each consumer household exists in a local neighborhood, buys products in local retail stores, and then uses those products. Buying products means selecting a store and then selecting a product from that store. Using products means selecting an item from inventory, reducing the stored amount of that item, and then updating the agent's memory about the product usage experience. These processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Each consumer household remembers varying attributes of both stores and products. The Miller (1956) range of seven plus or minus two bounds the sizes of the consumer attribute lists. The store and product selection processes depend on these attribute lists. These boundedly rational processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Households are randomly created based on draws from an input set of demographically indexed distributions. This process is $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Retail stores stock their shelves; adjust prices; compare their prices to other stores in their neighborhood; and offer promotions including flyers, sales, volume discounts, and in-store displays. Retail stores are initialized using an input list of properties. All of these boundedly rational processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Retail regions track sales volumes and profits at their stores and provide feedback to their stores on sales goals. Retail regions are initialized using an input list of properties. All of these boundedly rational processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Retailers track sales volumes and profits in their regions; provide feedback to their regions on sales goals; advertise; determine stocking options for regions and stores; and receive, use, and distribute trade support from manufacturers. Retailers are initialized using an input list of properties. All of these boundedly rational processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

Manufacturers maintain and update brand and product lists, advertise, run promotions, adjust suggested retail prices, and offer trade support incentives to retailers. Manufacturers are initialized using an input list of properties. All of these boundedly rational processes are $O(\lceil \log_2(N) \rceil)$ in time and space for each agent.

The agents in the model have a variety of attributes that can take on any double precision value as well as individualized lists, Boolean values, etc. As such, they are asymptotically incompressible.

The model produces two kinds of outputs. The first kind includes a result for each agent for each time step. The second kind consists of a summary line for each time step. There is a fixed number of each kind of output. Both types of outputs use straightforward data collection that is linearithmic or faster in time and space.

The input parameter ranges and counts do not increase with the number of consumers in the simulation, since the parameters provide constants for the previously described boundedly rational behavior. Therefore, modeling studies do not increase the asymptotic time or space complexity. This model implementation is thus $O(N\lceil \log_2(N) \rceil)$ in time and space.

This model fits into the requirements for result (14). Therefore, the Virtual Market Learning Lab is computationally optimal in asymptotic time and space performance.

Experimental results

For fixed word-size computers, we have $O(N\lceil \log_2(N) \rceil) = O(N)$, which is a linear relationship. As predicted, seven test measurements of the model execution time for a fixed number of time steps relative to N and a range of consumer population sizes were found to be collectively linear with an adjusted R^2 of 0.9963 and a p-value of 3.221×10^{-6}.

Conclusions and future work

Following Holland (1992, 1999, 2006), complex adaptive systems (CASs) are collections of interacting, autonomous, learning decision makers embedded in an interactive

environment. CASs are common in both nature and society. Modeling CASs is challenging for a variety of reasons. The challenges of modeling CASs can largely be overcome by using the individual-level focus of agent-based modeling. This paper's contribution is to introduce, analyze, and apply a theoretical formalism for proving findings about modular imperative agent-based models. The use of the formalism is demonstrated with three example models. These examples also show how the formalism is useful for predicting the execution time and space requirements for representations of common CASs. We have proven results (1) to (14), including that modular imperative agent-based modeling studies are asymptotically optimal in computational time and space for a common class of modeling problems. Given that modular imperative agent-based modeling is both computationally optimal and a natural structural match for many modeling problems, it follows that modular imperative agent-based modeling is the best modeling method for these situations.

There are many next steps for future work. First, the theoretical formalism can be used to analyze additional applied models for computational complexity and optimality. Second, the range of modeling problems covered by the computational optimality results in this paper might be expanded. Third, the performance results for modular imperative agent-based models violating the computational optimality criteria might be compared to that of other modeling techniques by directly using the formalism. Fourth, the formalism may be used to prove other properties of modular imperative agent-based models. Fifth, the formalism might be extended as modular imperative agent-based modeling practice evolves.

Endnote

[a]The term "linearithmic" refers to relationships of the form ($x \log_2 x$).

Abbreviations

ABM: Agent-based model; AC: Accumulator; CA-MRSA: Community-associated methicillin-resistant *Staphylococcus aureus*; CAS: Complex adaptive system; DEVS: Discrete Event System Specification; GB: Gigabyte; GHz: Gigahertz; GUI: Graphical user interface; HPC: High-performance computing; IC: Instruction counter; NP: Nondeterministic polynomial time; ODD: Overview, Design Concepts, and Details; P&G: Procter & Gamble; RAM: Random access machine; RASP: Random access stored-program; XML: Extensible markup language.

Competing interests

The author declares that he has no competing interests.

Acknowledgments

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science Laboratory, is operated under contract number DE-AC02-06CH11357. Part of this work was supported by National Science Foundation Division of Information and Intelligent Systems (IIS) award number 1125412. Part of this work was supported by National Institutes of Health National Institute of General Medical Sciences Models of Infectious Disease Agent Study (MIDAS) grant number U01GM087729. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the government.
The author wishes to thank Charles Macal, David Sallach, and William Rand for providing insightful reviews of this paper and perceptive suggestions for improvement. The author also wishes to thank the Repast team, the CA-MRSA modeling team, and the Virtual Market Learning Lab team for their contributions.
Any opinions, findings, conclusions, or recommendations are solely those of the author.

Received: 11 June 2013 Accepted: 7 March 2014
Published: 07 May 2014

References

Bolduc JS, Vangheluwe H: *A Modeling and Simulation Package for Classic Hierarchical DEVS.* Montreal, Quebec, Canada: Modelling, Simulation and Design Lab, School of Computer Science, McGill University, Technical Report; 2002.

Bonabeau E: **Agent-based modeling: Methods and techniques for simulating human systems**. In *Proceedings of the National Academy of Sciences*, 3, Volume 99. Washington, D.C. USA: National Academy of Sciences Press; 2002:7280–7287.

Brown DG, Riolo R, Robinson DT, North MJ, Rand W: **Spatial process and data models: toward integration of agent-based models and GIS.** *J Geogr Syst* 2005, **7**(1):25–47.

Christos CG, Lafortune S: *Introduction to Discrete Event Systems.* 2nd edition. New York, NY USA: Springer; 2008.

Collier NT, North MJ: **Parallel agent-based programming with Repast for High Performance Computing.** *Simulation* 2012:1–21.

Cook SA, Reckhow RA: **Time-bounded random access machines.** *J Comput Syst Sci* 1973, **7**(4):354–375.

Dorofeenko V, Shorish J: *Dynamical Modeling of the Demographic Prisoner's Dilemma.* Vienna, Austria: Institute for Advanced Studies Economics Series; 2002.

Elgot CC, Robinson A: **Random-access stored-program machines, an approach to programming languages.** *J ACM* 1964, **11**(4):365–399.

Epstein JM: **Zones of cooperation in demographic prisoner's dilemma.** *Complexity* 1998, **4**(2):36–48.

Epstein J: *Generative Social Science: Studies in Agent-Based Computational Modeling.* Princeton, NJ USA: Princeton University Press; 2007.

Foster I: *Designing and Building Parallel Programs.* Reading, MA USA: Addison-Wesley; 1995.

Grimm V, Berger U, Bastiansen F, Eliassen S, Ginot V, Giske J, Goss-Custard J, Grand T, Heinz SK, Huse G, Huth A, Jepsen JU, Jørgensen C, Mooij WM, Müller B, Pe'er G, Piou C, Railsback SF, Robbins AM, Robbins MM, Rossmanith E, Rüger N, Strand E, Souissi S, Stillman RA, Vabø R, Visser U, DeAngelis DL: **A standard protocol for describing individual-based and agent-based models.** *Ecol Model* 2006, **198**(1–2):115–126.

Hartmanis J: **Computational complexity of random access stored program machines.** *Math Syst Theory* 1970, **5**(3):232–245.

Harvey B: *Computer Science Logo Style.* Boston, MA USA: MIT Press; 1997.

Hinkelmann F, Murrugarra D, Jarrah AS, Laubenbacher RC: **A mathematical framework for agent based models of complex biological networks.** *Bull Math Biol* 2011, **73**(7):1583–1602.

Holland JH: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Cambridge, Mass: MIT Press; 1992.

Holland JH: *Emergence: From Chaos to Order.* Reading, Mass: Perseus Books; 1999.

Holland JH: **Studying complex adaptive systems.** *J Syst Sci Complex* 2006, **19:**1–8.

Kermack WO, McKendrick AG: **A contribution to the mathematical theory of epidemics.** *Proc Royal Soc A: Math, Phys Eng Sci* 1927, **15**(772):700.

Kleene SC: **Recursive predicates and quantifiers.** *Trans Am Math Soc* 1943, **53**(1):41–73.

Leombruni R, Richiardi M: **Why are economists skeptical about agent-based simulations?** *Phys A: Stat Mech Appl* 2005, **355:**103–109.

Lotka AJ: *Elements of Physical Biology.* Baltimore, Maryland, USA: Williams and Wilkins; 1925.

Luginbuhl DR: *Computational complexity of Random-Access Models. PhD thesis.* University of Illinois at Urbana-Champaign, Computer Science Department; 1970.

Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G: **MASON: A multiagent simulation environment.** *SIMULATION* 2005, **81:**517–527.

Macal CM: **To agent-based simulation from system dynamics**. In *Proceedings of the 2010 Winter Simulation Conference*. Edited by Johansson B, Jain S, Montoya-Torres J, Hugan J, Yücesan E. Baltimore, MD: IEEE/ACM; 2010.

Macal CM, North MJ: **Tutorial on agent-based modeling and simulation.** *J Simul* 2010, **4:**151–162.

Macal CM, North MJ, Collier N, Lauderdale DS, David MZ, Shumm P, Daum RS, Evans JA, Wilder JR, Dukic VM, Wegner DT: **Modeling the spread of community-associated MRSA**. In *Proceedings of the 2012 Winter Simulation Conference*. Edited by Laroque C, Himmelspach J, Pasupathy R, Rose O, Uhrmacher AM. Berlin: IEEE/ACM; 2012.

MASON: Accessed 2014 [http://cs.gmu.edu/~eclab/projects/mason]

Matsumoto M, Nishimura T: **Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator**. In *ACM Transactions on Modeling and Computer Simulation*, Volume 8. 1st edition. New York: ACM; 1998:3–30.

Miller GA: **The magic number seven plus or minus two.** *Psychol Rev* 1956, **63:**81–97.

Minar N, Burkhart R, Langton C, Askenazi M: *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations. Working Paper 96-06-042.* Santa Fe: Santa Fe Institute; 1996.

Murphy JT: *Computational social science and high performance computing: A case study of a simple model at large scales.* Santa Fe, NM USA: Proceedings of the 2011 Computational Social Science Society of the Americas Conference; 2011.

Myerson RB: *Game Theory: Analysis of Conflict.* Cambridge, MA USA: Harvard University Press; 1991.

NetLogo: Wilensky 2014 [http://ccl.northwestern.edu/netlogo/]

Niazi M, Hussain A: **A novel agent-based simulation framework for sensing in complex adaptive environments.** *IEEE Sensors J* 2010, **5**(3):1–9.

Nisan N, Roughgarden T, Tardos E, Vazirani VV: *Algorithmic Game Theory.* Cambridge, UK: Cambridge University Press; 2007.

North MJ: **A theoretical foundation for computational agent-based modeling**. *2012 World Congress on Social Simulation.* Taipei, Taiwan: National Chengchi University; 2012.

North MJ, Macal CM: *Managing Business Complexity: Discovering Strategic Solutions With Agent-Based Modeling and Simulation.* Oxford, U.K.: Oxford University Press; 2007.

North MJ, Macal CM: **Foundations of and recent advances in artificial life modeling with Repast 3 and Repast Simphony**. In *Artificial Life Models in Software.* 2nd edition. Edited by Adamatzky A, Komosinski M. Heidelberg: Springer; 2009:37–60.

North MJ, Collier NT, Vos RJ: **Experiences creating three implementations of the Repast agent modeling toolkit**. In *ACM Transactions on Modeling and Computer Simulation*, Volume 16. 1st edition. New York: ACM; 2006:1–25.

North MJ, Macal CM, St Aubin J, Thimmapuram P, Bragen M, Hahn J, Karr J, Brigham N, Lacy ME, Hampton D: **Multi-scale agent-based consumer market modeling.** *Complexity* 2010, **15**(5):37–47.

North MJ, Collier NT, Ozik J, Tatara E, Altaweel M, Macal CM, Bragen M, Sydelko P: **Complex adaptive systems modeling with Repast Simphony**. In *Complex Adaptive Systems Modeling*. Heidelberg: Springer; 2013.
Parker J, Epstein JM: **A distributed platform for global-scale agent-based models of disease transmission**. In *ACM Transactions on Modeling and Computer Simulation*, Volume 22. 1st edition. New York: ACM; 2011:1–25.
Parunak HVD, Savit R, Riolo RL: **Agent-based modeling vs. equation-based modeling: a case study and users' guide**. In *Proceedings of Multi-Agent Systems and Agent-Based Simulation*. New York, New York, USA: Springer; 1998:10–25.
Perumulla KS: **Parallel and distributed simulation: traditional techniques and recent advances**. In *Proceedings of the 2006 Winter Simulation Conference*. Edited by Perrone LF, Lawson B, Liu J, Wieland FP. Monterey. California: USA IEEE Press; 2006:84–95.
Rahmandad H, Sterman JD: **Heterogeneity and network structure in the dynamics of diffusion: comparing agent-based and differential equation models.** *Manag Sci* 2008, **54**(5):998–1014.
Repast 3: Accessed 2014 [http://repast.sourceforge.net/repast_3]
Repast for High Performance Computing: Accessed 2014 [http://repast.sourceforge.net/repast_hpc.html]
Repast Simphony: Accessed 2014 [http://repast.sourceforge.net/repast_simphony.html]
Resnick M: *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA, USA: MIT Press; 1994.
Shang H, Wainer G: **A simulation algorithm for dynamic structure DEVS modeling**. In *Proceedings of the 2006 Winter Simulation Conference*. Edited by Perrone LF, Wieland FP, Liu L, Lawson BG, Nicol DM, Fujimoto RM. Monterey, CA, USA: IEEE Press; 2006:815–822.
Smale S: **Differentiable dynamical systems.** *Bull Am Math Soc* 1967, **73**(6):747–817.
Spivey JM: *The Z Notation: A Reference Manual*. 2nd edition. Oxford, UK: Programming Research Group, University of Oxford; 1992.
StarLogo: Accessed 2014 [http://education.mit.edu/starlogo/]
Swarm: Accessed 2014 [http://www.swarm.org/]
Volterra V: **Variazioni e fluttuazioni del numero d'individui in specie animali conviventi.** *Mem. Acad. Lincei Roma* 1926, **2**:31–113.
Wilensky U: *NetLogo*. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University; 1999.
Wilson WG: **Resolving discrepancies between deterministic population models and individual-based simulations.** *Am Nat* 1998, **151**(2):116–134.
Zeigler BP, Praehofer H, Kim TG: *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2nd edition. San Diego, CA USA: Academic Press; 2000.

10.1186/2194-3206-2-3
Cite this article as: North: A theoretical formalism for analyzing agent-based models. *Complex Adaptive Systems Modeling* 2014, 2:3